計算機アーキテクチャ　80－7
（1990．1．25）

SIMP（単一命令流／多重命令パイプライン）方式に基づくスーパースカラ・プロセッサ『新風』の命令供給機構

原　哲也　久我守弘　村上和彰　富田眞治
（九州大学大学院総合理工学研究科）

我々は単一プロセッサの処理能力向上を目的として，従来の命令パイプライン処理と低レベル（命令レベル）並列処理とを融合した，SIMP（Single Instruction Stream／Multiple Instruction Pipelining：単一命令流／多重命令パイプライン）方式を提案し，この方式に基づく試作機としてスーパースカラ・プロセッサ『新風』を開発中である．『新風』プロセッサは，4本の命令パイプラインによって単一命令流の並列処理を行う．このため，パイプラインの乱れによる性能低下は単一命令パイプラインの時よりもさらに深刻になる．命令供給機構では，パイプラインの乱れを起こす要因の中で，命令間の制御依存関係と命令供給の遅れに対処し，命令不在によるパイプラインの乱れを抑える役割を担っている．この要求に対し以下の対策を行った．

4本の命令パイプラインの処理能力に応じた命令供給を行うため，4つの命令を同時にフェッチして命令ブロックを作成し命令パイプラインに供給する．また，分岐命令に起因する制御依存関係に対処するためにBTB方式による分岐予測を行っている．この場合，誤った分岐予測により命令パイプラインに投入された実行すべきでない命令を無効化する必要がある．このとき，『新風』では命令パイプラインに投入されている命令数が多い（最大23命令）ことを考慮して，実行してはならない命令のみを選択的に無効化する“選択的命令無効化”を採用した．

本稿では，命令供給機構の設計方針，その構成および動作，性能について述べる．

Instruction Supply Mechanism in the SIMP Processor Prototype

(in japanese)

Tetsuya HARA, Morihiro KUGA, Kazuaki MURAKAMI, and Shinji TOMITA
Department of Information Systems
Interdisciplinary Graduate School of Engineering Sciences
Kyushu University
6-1, Kasuga-koen, Kasuga-shi, Fukuoka, 816 Japan
E-mail : hara@kyu-is.is.kyushu-u.ac.jp

SIMP (Single Instruction Stream／Multiple Instruction Pipelining) is a multiple-instruction pipeline parallel architecture which combines instruction pipelining and low-level parallel processing. We are developing the SIMP processor prototype :『新風』(in Japanese). The SIMP processor prototype provides 4 instruction pipelines, and executes a single instruction stream with parallel processing. The detrimental effects to performance caused by branch instruction, therefore, are more severe than those of the single-instruction pipeline processor.

The instruction supply mechanism copes with control dependencies and delay of supplying instruction. To accomplish this, the instruction supply mechanism resolves ploblems as follows.

(i) Every cycle, to supply instructions according to ability of the IPUs' (Instruction Pipeline Units) preformance, the IBSU (Instruction-BlickSupply Unit) fetches 4 successive instructions from the instruction cache at the same time and constructs an instruction block, then supplies it to the IPUs.

(ii) To resolve control dependencies which are caused by branch instructions, the instruction block to be prefetched is determined with the help of branch prediction with a Branch Target Buffer.

(iii) As the prediction-miss handling techniques, the IBSU provides the selective instruction-squashing scheme, which squashes only the instructions to be excluded from the dynamic instruction stream. It is because, there is a good possibility that the instructions to be refetched already exist in the IPUs even though the prediction was miss.

In this paper, we describe the design philosophy and the hardware mechanism of the instruction supply unit, and its performance estimate.

1. はじめに

我々は単一プロセッサの処理能力向上を目的として，従来の命令パイプライン処理と低レベル（命令レベル）並列処理とを融合した，SIMP（Single Instruction Stream／Multiple Instruction Pipelining：単一命令流／多重命令パイプライン）方式を提案し，[1] [3] この方式に基づく試作機としてスーパースカラ・プロセッサ『新風』を開発中である．[2] [4]

『新風』プロセッサは，4本の命令パイプラインによって単一命令流の並列処理を行う．このため，パイプラインの乱れによる性能低下は，単一命令パイプラインの時よりもさらに深刻なものとなる．パイプラインの乱れを起こす要因としては，命令間のデータ依存，分岐命令に起因する制御依存，命令やデータ供給の遅れ，および，演算機能の時間的なバラツキなどが挙げられる．

命令供給機構では，これらの要因の中でも，制御依存と命令供給の遅れに対処し，命令不在によるパイプラインの乱れを抑える役割を担っている．

本稿では，命令供給機構に対する要件と設計方針をまとめたあと，その構成および動作，シミュレーションによる性能予測について述べる．

2. 要件

命令供給機構は，命令不在が生じないように有効な命令を常時命令パイプラインに供給する必要がある．これには，

① 命令パイプラインの処理能力に応じた命令の供給
② 分岐命令に起因する制御依存への対処
③ 分岐命令実行後の効率的な復元処理

などが課題となる．これらの課題に対処するため，以下に示す3つの機能が必要となる．[5]

(1) 命令プリフェッチ

命令パイプラインの処理能力に応じた命令の供給を行う．『新風』プロセッサでは，4本の命令パイプライン（IPU：Instruction Pipeline Unit）を用いて並列処理を行っているため，パイプライン・サイクル（120ns）毎に4個の命令を同時に処理可能である．したがって，4個の命令を毎サイクル同時に命令キャッシュからフェッチし，命令パイプラインに供給しなければならない（これら4個の命令を“命令ブロック”と呼ぶ）．

(2) 分岐予測

分岐命令に起因する制御依存への対処を行う．分岐命令には，無条件分岐命令と条件分岐命令とがある．条件分岐命令の場合，分岐するか否か（TAKEN／NOT-TAKEN）は命令を実行してみないと決まらない．また，いずれの命令でも，分岐先アドレスは分岐命令の実行により一般に決まるので，それまで次にフェッチすべき命令のアドレスが定まらない．このとき，分岐命令のTAKEN／NOT-TAKENおよび分岐先アドレスが確定するまで命令フェッチを止めてしまうと，命令不在が生じパイプラインに空きが生じる．そこで，このような命令不在を避けるためには何らかの分岐予測を行い，その予測結果に従って命令をパイプラインに投入する必要がある．

(3) パイプライン復元処理

分岐命令実行後，命令パイプラインの効率的な復元処理（repair）を行う．分岐予測を行っている場合，当該分岐命令が実行された後，予測が正しかったか否かを調べる．もし予測が誤りの場合，パイプラインに既に投入されている命令を無効化する必要がある．

3. 設計方針

従来の単一命令パイプラインにおいては，命令不在によるパイプラインの乱れを抑えるために様々な手法を用いている．[11] ~ [20] 『新風』においても，これらの手法は有効である．さらに，『新風』特有の要件を満たすため，2章で述べた課題を解決する必要がある．以下，従来手法も含めた様々な解決案の選択肢を示しながら，『新風』の命令供給機構の設計方針を述べる．

3.1 命令プリフェッチ

まず，命令キャッシュから4個の命令を同時に読み出せるように，そのデータ巾を128-bit（＝32-bit／命令×4命令）とする．これにより，毎サイクル4命令を同時にフェッチして，それらを命令ブロックとしてIPUに投入するが可能となる．

しかしここで，命令ブロック内でのこれら4命令の並びの順序，すなわち，命令とIPUとの対応関係が問題となる．[1] これは，1命令ブロック内の命令の実行順序が，命令の投入されたIPUの番号（若い方が先）で決まるからである．よって，命令ブロック内の命令の並び換え（命令アラインメント）を行うか否かに関して，次の2つの選択肢がある．

① 命令アラインメント無し：命令キャッシュから読み出した4命令をそのまま，その順序で各IPUに投入する．命令キャッシュのバンク分割およびアラインメント回路が不要であり，ハードウェアが簡単になる．しかし，分岐の結果，分岐先命令が命令ブロックの先頭に来なかった場合，それ以前の命令をno-opにする必要がある．このno-op命令により，パイプライン使用率が下がり性能が低下する恐れがある．

② 命令アラインメント有り：任意のワードアドレスから連続する4命令を読み出せるよう，命令キャッシュを4バンク（32-bit巾／バンク）に分割する．そして，実行順序の最も早い命令が命令ブロックの先頭となるように，命令のアラインメントを行う．この場合，①と異なり，無駄なno-op命令が命令ブロックに挿入されないので，パイプライン使用率は良い．ただし，命令キャッシュのバンク分割およびアラインメント回路が必要となる．

これに関しては，性能を重視して②を選択する．4バンク分割した命令キャッシュをマルチバンク命令キャッシュ（MBIC：Multiple-Bank Instruction Cache）と呼ぶ．ただし，静的コード・スケジューリングの観点からは，動的な命令アラインメントは逆に静的コード配置を難しくする要因となる．[8] [9] そこで，命令アラインメントを行わないモードも設けている．

さて，命令アラインメントを行う場合，命令ブロックはキャッシュ・ライン内の任意のワードアドレスから連続する4命令で構成されるので，ラインをまたぐ（ラインクロス）可能性が出て来る．すなわち，1命令ブロックを構成する4命令が連続する2個のラインにまたがる可能性がある．よって，このラインクロスを許すか否かに関して，次の2つの選択肢がある．

a）ラインクロス不可：次のラインに存在する命令の読出しを行わないので，命令ブロック内の相当する部分にno-op命令を挿入しなければならない．よって，パイプライン使用率が下がる恐れがある．しかしながら，i）1パイプライン・サイクル中に2度命令キャッシュにアクセスする必要がない，ii）パイプライン内に存在する命令のタグ情報（命令アドレスなど）の管理が容易となる，といった長所がある．

b）ラインクロス可：aに比べると，パイプライン使用率は良い．しかし，aの長所が得られず，その分ハードウェアが複雑になる欠点がある．特に，1パイプライン・サイクル中に2度も命令キャッシュにアクセスするのは，極めて困難である．しかも，2番目のラインがミスヒットであれば，性能はaと変わらない．

これに関しては，ハードウェアの簡易性を重視してaの方針を採る．なお，パイプライン使用率，すなわち，1命令ブロック中のno-opの割合については，以下の考察が成り立つ．

① 命令アラインメント無し：1命令ブロック中，平均1.5命令がno-opとなる．

②-a 命令アラインメント有り＆ラインクロス不可：1ライン中，平均1.5命令がno-opとなる．よって，1命令ブロック中，

平均0.375命令がno-opとなる.

②-b 命令アラインメント有り＆ラインクロス可：2番目のラインが必ずヒットすれば，no-opになる命令は0である.

これから，命令アラインメントの有無がパイプライン使用率に与える影響は大きいものの，ラインクロスの可／不可はさほど影響を与えないことが予想される.

3.2 分岐予測

分岐命令に起因する制御依存への対処法として，以下のように種々の方式が提案されている.[13] [15] [18]

① 分岐アーキテクチャ自身による対処：先行条件決定方式（advanced conditioning），[7] [11] 分岐予告方式（prepare-to-branch instruction），[18] など.

② 分岐命令自身による対処：遅延分岐命令，静的分岐予測を行う条件分岐命令，など.[14] [17]

③ 分岐命令実行に関する対処：条件分岐命令実行における早期分岐解消（early branch resolution），[18] 分岐命令畳み込み（branch folding），[15] など.

④ 命令供給に関する対処：動的分岐予測，[12] ~ [14] 分岐パイパス（branch bypassing）／複数命令プリフェッチ（multiple prefetching），[13] [18] など.

これらの対処法はほぼ直交関係にあり，様々な組合せが可能である.『新風』では，①については先行条件決定方式（advanced conditioning）を，[7] ③については早期分岐解消（early branch resolution）を，④については動的分岐予測を，[5] [6] 採用している.以下，命令供給機構に関係のある動的分岐予測について考える.

さて，動的分岐予測1つをとっても，様々な手法がある.[12] ~ [14]『新風』では，その中でも，精度の高い分岐予測が可能なBTB（Branch Target Buffer）方式 [13] を採用する.BTBの構成法および使用法にも，後述するように様々ある.基本的には，過去に実行した分岐命令の命令アドレス，その分岐結果（TAKEN／NOT-TAKEN）の履歴，および，分岐先の命令アドレス（または命令自身）を保持する複数のエントリから構成される.その使用法は，i) 分岐命令をフェッチ（あるいはデコード）するとBTBを検索し，当該分岐命令が登録されているか否かチェックする，ii) もし登録されていれば，その履歴に基づき分岐するか否かを予測し，予測した側の命令をプリフェッチする，となる.

BTBの構成法および使用法に関する選択肢を以下にまとめ，『新風』で採った方針を示す.

(1) BTBの検索時期および配置場所

まず，分岐予測をどのステージで行うか，すなわち，BTBの検索時期と配置場所に関して，次の2つの選択肢がある.

① 命令フェッチ・ステージ：命令キャッシュにアクセスすると同時に，BTBを検索して次サイクルのフェッチ・アドレスを決定する.この場合，BTBにも命令キャッシュと同程度のアクセス速度が要求され，BTBも命令キャッシュの近くに配置する.

② デコード・ステージ以降：命令をデコードして分岐命令を検出した後の適当な時期に，BTBを検索する.よって，BTBは任意の場所に配置され，アクセス速度にもさほど高速性が要求されない.しかし，①に比べるとBTBを用いた分岐予測の時期が遅いので，無駄な命令がフェッチされる可能性がある.

これに関しては，性能を重視して①を採り，MBIC内にBTBを設ける.

(2) BTBの検索法

BTBはその機能上，一種の連想メモリとして動作する.上の(1)で命令キャッシュ近傍にBTBを配置すると，命令キャッシュ自身も連想メモリであることから，互いの連想性に関して次の2つの選択肢が生じる.

① 命令キャッシュから独立：命令キャッシュの連想性とは無関係に，BTBの連想性を定める.理想的には，BTBをフルアソシアティブにすることも可能であるが，ハードウェア量の増加を招く.

② 命令キャッシュと連動：命令キャッシュの連想性をBTBが共用する.この場合，命令キャッシュとBTBに対してただ1個のタグ・アレイしか存在しないので，両者は連動することになる.[19]

これに関しては，ハードウェアの簡易性から②の方針を採る.MBICはダイレクト・マッピング方式であり，そのタグ・アレイはBTBとの間で共有される.よって，MBICへのアクセスと同時にBTBにもアクセスすることになる.

(3) BTBエントリ数

上の(2)で命令キャッシュとBTBとを連動させると，BTBのエントリ数は命令キャッシュのライン数で決まる.このとき，1ライン当り何個のBTBエントリを設けるか，つまり，1ライン当りの命令数（『新風』では16命令）とBTBエントリ数の最適な比率が問題となる（『新風』では，BTBエントリ数が4で4：1の比率である）.さらに，BTBエントリが足りなくなった場合の対策として，BTBエントリのリプレースメントを行うか否かが問題となる（『新風』では行わない）.これらについては，5.2節および5.3節でそれぞれ検討する.

(4) BTBエントリの登録情報

BTBエントリに登録する分岐先に関する情報として，

① 分岐先命令アドレス

② 分岐先命令

の2つの選択肢がある.これに関しては，①を選択する.これは，BTBが命令キャッシュと連動し，かつ，命令キャッシュが大容量（512Kバイト）であることから，分岐先アドレスさえ得られれば直ちに（正確には次サイクルで）分岐先命令をフェッチできるからである.

(5) 分岐予測アルゴリズム

分岐予測の精度を高めるには，BTBエントリに保持する分岐結果履歴のビット数，および，履歴に基づく分岐予測アルゴリズムが問題となる.まず，履歴ビット数については，参考文献[13]等に報告されているように，2-bitで予測のヒット率が飽和傾向を示し始めるので，2-bitで十分と判断した.2-bitの履歴に基づく分岐予測アルゴリズムについては，5.1節で検討する.

(6) 分岐予測の対象

これはSIMP方式特有の課題である.[1] すなわち，複数命令を同時にフェッチし，それらを命令ブロックとしてパイプライン処理するので，

① 命令ブロック内の個々の命令に対して分岐予測を行うか

② 1個の命令ブロック全体に対して分岐予測を行うか

が問題となる.分岐予測の精度の点からは①が望ましいが，その処理は本質的に逐次処理であり，これがクリティカル・パスとなる恐れがある.一方，②では，1命令ブロック内に複数の分岐命令が含まれる場合，ある分岐命令の予測が命令ブロック全体の予測に反映されない可能性がある.その意味で，①より予測精度は落ちるが，簡単なハードウェアで直ちに予測を行える.『新風』では，②の方針を採っている.

3.3 パイプライン復元処理

一般に分岐予測を行う場合，その予測がはずれた際，誤って命令パイプラインに投入された命令を無効化してパイプラインを復元する必要がある.この復元処理の方法には，次の2つの選択肢が存在する.

① パイプライン・フラッシュ（pipeline flush）：分岐予測をはずした分岐命令以降のすべての命令を単に無効化する.そして，当該分岐命令の分岐結果に従って，正しい命令を再フェッチする.

② 選択的命令無効化（selective instruction

squashing)[1]：無効化すべき命令を選択し，それのみを無効化する．もし有効な命令が残らなかった場合にのみ，命令の再フェッチを行う．

これに関しては，以下の2つの理由により②の方針を採る．

a)『新風』では，当該分岐命令以降に命令パイプラインに投入され得る命令の数が最大23命令と多く，分岐予測がはずれていても正しい命令が含まれている可能性がある．よって，命令パイプライン内の命令を有効に利用するため，選択的命令無効化を採用する．

b) 命令ブロック全体を分岐予測の対象としているため（3.2節参照），分岐予測が当たっても命令の無効化が必要となる可能性がある．たとえば，命令ブロックの最終命令以外が分岐命令で，かつ，結果がtakenの場合，予測がtakenと当たっていても，その命令ブロックの当該分岐命令以降の命令は無効化されねばならない．よって，パイプライン復元処理以外の目的でも，選択的命令無効化の機能が必要となる．

4. 構成と動作

4.1 全体構成

命令供給機構の全体構成を図1に示す．命令供給機構は，以下の2つの部分から構成される．

(1) マルチバンク命令キャッシュ（MBIC）

図2に示すように，MBICは次の4つから構成される．

① タグ・アレイ：ダイレクト・マッピング方式で，仮想アドレスによりアクセスされる（仮想アドレス・キャッシュ）．ライン・サイズは，64バイト（＝16命令）である．

② 命令アレイ：連続する4命令を任意のワードアドレスから同時にフェッチできるように，4バンク（32-bit／バンク）構成とする．キャッシュ容量は512Kバイトである．

③ 命令アラインメント回路：命令アレイからフェッチした4命令に対して，最も若番の命令が先頭に来るように並び換える．

④ BTB (Branch Target Buffer)：1ライン当り，4個のBTBエントリを設ける．命令アレイとの間で，タグ・アレイを共有する．

(2) 命令ブロック供給機構（IBSU）

IBSUは4本のIPUに共有され，命令パイプラインにおける最初のIF（命令ブロック・フェッチ）ステージを担当する．図1に示すように，次の3つの主要機構と2つの資源とから構成される．

① 命令プリフェッチ機構：命令ブロックのプリフェッチを行う（4.2節参照）．

② 分岐予測機構：BTBを用いた分岐予測を行う（4.3節参照）．

③ 命令再フェッチ機構：選択的命令無効化によるパイプライン復元処理を行う（4. 4節参照）．

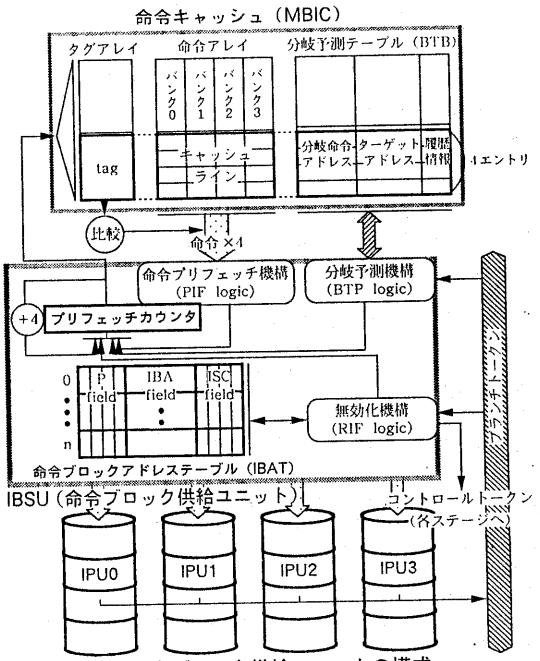

図1. 命令ブロック供給ユニットの構成

④ プリフェッチ・カウンタ（PFC：PreFetch Counter）：プリフェッチすべき命令ブロックの先頭ワードアドレス（SWA：Starting Word Address）を保持する．

⑤ 命令ブロック・アドレス・テーブル（IBAT：Instruction-Block Address Table）：現在IPU中に存在している命令ブロックのタグ情報の一部を管理するテーブルである．IPUには最大7命令ブロックが存在し得ることから，7エントリのリングテーブル構成となっている．各エントリは，次のフィールドを有する．

- IBA (Instruction Block Address)：命令ブロックの先頭命令のワードアドレス（30-bit）
- NIBA (Next Instruction Block Address)：プログラムの字面上で，次の命令ブロックの先頭命令に相当する命令のワードアドレス（NIBA＝IBA＋4；30-bit）
- P (Prediction)：命令ブロック内のどの命令に対して分岐予測を行ったかを示す4ビット
- ISC (Instruction Squash Control)：命令ブロック内のど

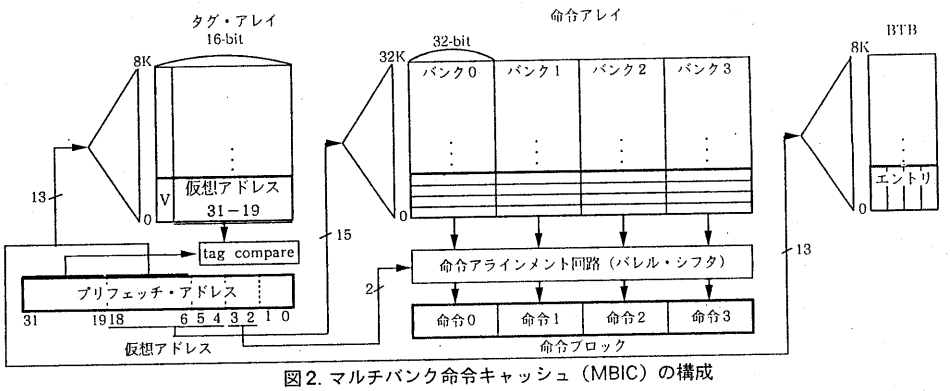

図2. マルチバンク命令キャッシュ（MBIC）の構成

の命令が無効化されているかを示す4ビット

4.2 命令プリフェッチ機構

命令プリフェッチ機構は以下の手順で，命令ブロックをプリフェッチする．

(1) MBICからの命令読出し

PFC内のSWAを用いて，MBICの命令アレイから連続する4命令を読み出す．SWAは命令ブロックの先頭命令のワードアドレスであるので，SWA（正確には，そのセットアドレス部分）にそれぞれ+1，+2，+3の加算を行い後続命令のバンク内ワードアドレスを求める．このとき，ラインクロスの発生（ライン内アドレス部分からの桁上がり）を検出する．もしラインクロスする命令があれば，当該命令をno-opに置き換える．

(2) 命令ブロックの作成

命令アレイから読み出した命令に対して，SWAで示される命令が命令ブロックの先頭に来るようにアラインメントを行う．そして，その結果を命令ブロックとして，各命令を対応するIPUに投入する．

(3) IBATへの登録

命令ブロックを作成したら，以下のように，当該命令ブロックに関する情報をIBATに登録する．

① IBAフィールド：SWAを登録
② NIBAフィールド：SWA+4を登録
③ Pフィールド：分岐予測の結果takenと予測した場合，当該分岐命令のPビットを1に設定
④ ISCフィールド：ラインクロスによりno-opに置き換えられた命令のISCビットを1に設定

(4) プリフェッチ・カウンタの更新

次サイクルでプリフェッチすべき命令ブロックの先頭命令のワードアドレス（次SWA）を以下の中から決定する．

① 連続アドレス：通常の連続フェッチにおいては，現SWAに+4の加算を行って次SWAとする．ただし，ラインクロスを起こした場合は，次SWAの下位4ビットを全て0にしてライン先頭アドレスとする．
② 予測分岐先アドレス（PTA：Predicted Target Address）：分岐予測の結果takenと予測された場合，分岐予測機構から送られて来るPTAを次SWAとする．
③ 再フェッチ・アドレス（RFA：ReFetch Address）：分岐命令の実行結果，命令再フェッチが必要となった場合，命令再フェッチ機構から送られて来るRFAを次SWAとする．

これらの優先順位は，高い方から③→②→①の順である．

4.3 分岐予測機構

分岐予測機構は，BTBを用いた分岐予測，および，BTBの登録・更新を行う．

(1) BTBの構成

BTBの構成を図3に示す．命令キャッシュの1ライン当り4個のBTBエントリが対応しており，これらは同時に読み出せる．各エントリは，以下の4つのフィールドを有する．

① V（Valid）：エントリの有効／無効
② BIA（Branch Instruction Address）：分岐命令アドレスのライン内アドレス部分（4-bit）
③ PTA（Predicted Target Address）：分岐先命令のワードアドレス（30-bit）
④ 履歴（History）：過去の分岐結果（TAKEN／NOT-TAKEN）の履歴（2-bit）

(2) 分岐予測処理

命令アレイからの4命令読出しと並行して，以下の手順で分岐予測を行う．

① BTB検索：命令読出しに用いている4個のワードアドレス（SWA～SWA+3）の下位4ビットと，当該ラインに対応する4個のBTBエントリのBIAとを総当たりで比較する（16個の4ビット長比較器を使用）．

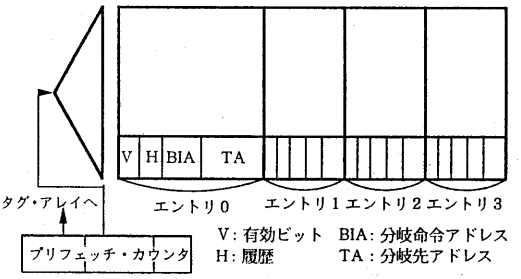

図3. 分岐ターゲット・バッファ（BTB）の構成

② 分岐予測：もし一致する有効なエントリが存在すれば，当該命令は分岐命令である．その履歴を基に5.1節で述べる分岐予測アルゴリズムに従って，分岐予測を行う．
③ 分岐予測の結果takenと予測した場合は，当該BTBエントリのPTA（予測分岐先アドレス）を命令プリフェッチ機構に送る．

ここで，tatenと予測された分岐命令が複数あれば，最も若番の分岐命令を選択する．これは，最も早く実行される分岐命令に対して，それ以降の分岐命令が制御依存関係にあるからである．

(3) BTBの登録・更新処理

BTBの登録・更新は，分岐命令の実行結果に基づいて行う．分岐命令の実行結果は，IPUの実行ステージからIBSUへ分岐トークンにより送られる．[4] 分岐トークンは，以下の3つのフィールドを有する．

- BID（Branch Instruction ID）：当該分岐命令の命令識別子（IBATエントリ番号）
- R（Result）：分岐結果（TAKEN／NOT-TAKEN）
- TA（Target Address）：分岐結果がTAKENの場合の分岐先命令アドレス

BTBの登録・更新は，以下の手順で行う．

① 分岐トークンが受け取ると，まず，そのBIDを用いてIBATエントリにアクセスし，当該分岐命令の命令アドレスを求める．
② 次に，当該分岐命令がBTBに登録されているかを調べる．
③ すでに登録されている場合，分岐トークンのRにより当該BTBエントリの履歴を更新するとともに，TAKENであれば当該BTBエントリのTAを上書きする．
④ 登録されていない場合，分岐結果がTAKENであれば空きBTBエントリに登録する．もし空きBTBエントリがなければ登録しない．また，分岐結果がNOT-TAKENであれば，空きBTBエントリがあっても登録しない．

4.4 命令再フェッチ機構

命令再フェッチ機構は，分岐命令が実行される度に，命令パイプライン内の不要命令を選択して無効化する．もし有効な命令が存在しない場合には，命令再フェッチを行う．

選択的命令無効化は，分岐トークンを基にIBATを探索し，後続命令中に目的とする命令が存在するか否かをチェックすることで実現する．このとき，分岐予測（taken／not-taken）と分岐結果（TAKEN／NOT-TAKEN）の組合せ（「予測→結果」で表記）によって，以下の3通りの場合が存在する．[3] [5]

(1) not-taken→NOT-TAKEN（図4（a））の場合

命令パイプライン内の命令はすべて有効なので，無効化を行う必要はない．

(2) taken→TAKEN（図4（b））またはnot-taken→TAKEN（図4（c））の場合

分岐結果がTAKENである場合，その分岐先命令がすでに命令パイプラインに投入されているかどうかの探索を行う．探索に用いる「探索アドレス」は，分岐トークン中のTA（分岐先ア

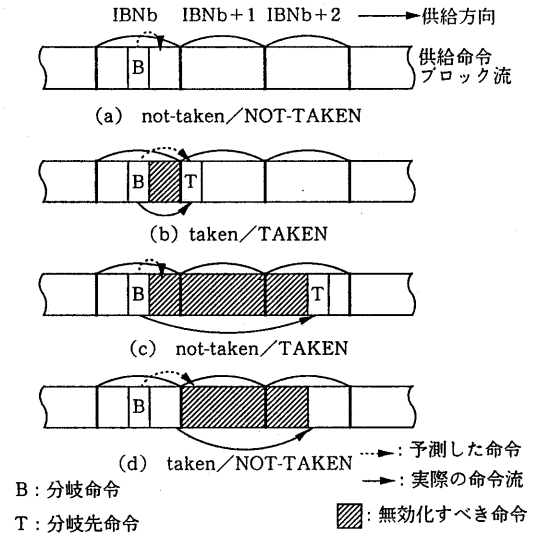

図4. 選択的命令無効化

ドレス）である．探索は，分岐トークン中のBID（分岐命令ID）で示されるIBATエントリから最尾エントリまでの範囲で，「分岐予測が到達する命令のパス（後述）」に限定して行う．もし分岐先命令が見つかれば，分岐命令の次から分岐先命令の手前までを無効化する．そうでなければ，分岐命令以降のすべてを無効化し，TAをRFA（再フェッチ・アドレス）として命令プリフェッチ機構に送る．

(3) taken→NOT-TAKEN（図4 (d)）の場合

プログラムの字面上，次の命令ブロックに相当する命令ブロックの先頭命令を探索する．探索に用いる「探索アドレス」は，分岐トークン中のBID（分岐命令ID）で示されるIBATエントリのNIBAフィールドから得る．これ以降の処理は，bと同様である．

さて，「分岐予測が到達する命令のパス」を定義する．3.2節で述べたように，命令ブロック単位での分岐予測を行なっているため，IPUに供給される一連の命令ブロック（供給命令ブロック流と呼ぶ）には，図5に示すように次の2種類の命令が混在する．[10]

① 分岐予測が到達する命令：供給命令ブロック流の先頭から始めて，分岐予測に従って順に命令をトラバースしていって到達する命令．トラバースした経路を「分岐予測が到達する命令のパス」と呼ぶ．

② 分岐予測が到達しない命令：①以外の命令．

先の(2)(3)における探索で，「分岐予測が到達しない命令」をその探索対象に含んだ場合の問題点は図5から明らかである．

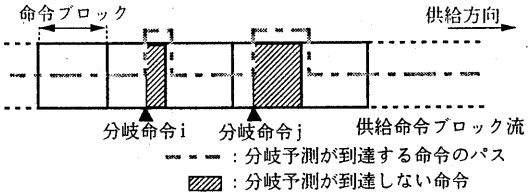

図5. 供給命令ブロック流と予測命令パス

5. 分岐予測の仕様および評価

分岐予測アルゴリズム，BTBエントリ数，および，BTBエントリのリプレースメントについて，トレース駆動シミュレーションによる評価を行い仕様を決定した．

5.1 分岐予測アルゴリズム

2-bitの履歴に基づく分岐予測アルゴリズムとして，図6に示す4状態遷移を行う以下の3種類のアルゴリズムを検討する．[6]

① Φ型分岐予測アルゴリズム（図6 (a)）
② Z型分岐予測アルゴリズム（図6 (b)）
③ O型分岐予測アルゴリズム（図6 (c)）

なお，状態遷移図で状態を示すノード内の小文字が分岐予測を表わし，tならばtaken，nならばnot-takenと予測する．また，状態遷移を示す有向辺上の大文字が分岐結果を表わし，TAKENならT，NOT-TAKENならNと記された有向辺に従って状態を遷移させる．ちなみに，上記の3種類の分岐予測アルゴリズムの型名は，状態遷移図の形状を表わしている．

2-bitの履歴に基づくことから4つの状態が存在するが，このうちどれをBTBエントリ登録の際の初期状態にするかについて考察する．まず，状態 '00' と状態 '11' はともに安定状態であるので初期状態の候補から外す．残りの状態 '01' および状態 '10' のいずれかを初期状態とした場合の予測ミスヒット率を，代表的な分岐パターンの発生頻度を基に算出した．その結果，状態 '10' を初期状態とした方が予測ミスヒット率が低いことが判明している．[6] [10]

トレース駆動シミュレーションを用いて，3種類の分岐予測アルゴリズムによる予測ヒット率を評価した．トレースデータは，Sun4 (SPARC) 上でアッカーマン関数，バブルソート，Cコンパイラ，Dhrystone，Yaccを実行して採集したものである．なお，以下をシミュレーションの前提とした．

- 初期状態：状態 '10'
- BTBエントリ数：4エントリ／ライン
- BTBエントリのリプレースメント：行わない

シミュレーション結果を表1に示す．表1より，分岐予測アルゴリズムによるヒット率の違いはほとんどないことがわかる．ただ，O型分岐予測アルゴリズムが平均して最も予測ヒット率が高く，かつ，どのベンチマークに対しても一定して高い予測ヒット率を得ている．よって，O型分岐予測アルゴリズムを採用することにし，以下の議論もO型分岐予測アルゴリズムを前提にするものとする．

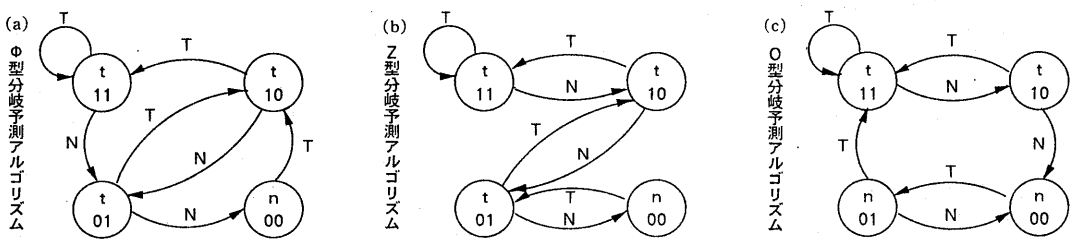

図6. 分岐予測アルゴリズム

表1. 各予測アルゴリズムのヒット率（%）

	予測アルゴリズム		
ベンチマーク	Z型	Φ型	O型
アッカーマン関数	99.69	99.68	99.68
バブルソート	94.09	94.14	94.16
Cコンパイラ	88.57	88.21	88.52
dhrystone	88.91	87.71	90.09
yacc	88.07	87.68	88.16
平均	91.86	91.48	92.12

5.2 BTBエントリ数

3.2節で述べたように，1ライン当りに何個のBTBエントリを設けるか，つまり，1ライン当りの命令数（『新風』では16命令）とBTBエントリ数の最適な比率が問題である．分岐命令の出現頻度は20%強と一般に言われており，約4命令に1命令が分岐命令ということになる．すなわち，BTBエントリを4個設ければよいと考えられる（『新風』では，BTBエントリ数が4で4：1の比率である）．

そこで，5. 1節と同一のトレース駆動シミュレーションにより，1ライン当りのBTBエントリ数を3, 4, 5および16（理想的な場合）としたときの予測ヒット率を調べた．以下をシミュレーションの前提とした．

- 分岐予測アルゴリズム：O型
- 初期状態：状態 '10'
- BTBエントリのリプレースメント：行わない

シミュレーション結果を表2に示す．表2より，BTBエントリを4個以上としても，予測ヒット率にほとんど向上が見られないことがわかる．よって，『新風』の現仕様である4個／ラインのBTBエントリ数は極めて妥当であると言える．

表2.エントリ数によるヒット率（%）

エントリ数	3	4置換あり	4	5	16
ヒット率（%）	88.35	92.30	92.12	92.52	92.53

5.3 BTBエントリ置換

1ライン当り4個のBTBエントリを設けても，命令の偏りによって1ライン内にTAKENとなる分岐命令が5個以上存在することもあり得る．このとき，BTBエントリのリプレースメントが必要か否かを検討した．

トレース駆動シミュレーションは，5.1節と同一である．以下をシミュレーションの前提とした．

- 分岐予測アルゴリズム：O型
- 初期状態：状態 '10'
- BTBエントリ数：4エントリ／ライン
- リプレースメント対象となる状態の優先順位：状態 '00' → 状態 '01' →状態 '10' →状態 '11'

シミュレーション結果を表2に示す．表2より，リプレースメントの有無による予測ヒット率の違いがほとんどないことがわかる．よって，BTBエントリのリプレースメントは行わない方針とした．

6. 性能予測

命令供給機構の性能を予測するため，『新風』シミュレータ [2] を用いて，以下の項目に関する性能評価を行った．

① 命令アラインメント：有／無．命令アラインメント有りについては，さらに，

①' ラインクロス：可／不可

② 分岐予測：有／無．分岐予測有りについては，さらに，

②' 履歴：1-bit／2-bit

③ 選択的命令無効化：有／無（パイプライン・フラッシュ）

なお，以下をすべてのシミュレーションを通しての共通前提とした．

- BTBエントリ数：4エントリ／ライン
- BTBエントリのリプレースメント：行わない
- 命令キャッシュおよびデータキャッシュのミスヒット：無し

また，ベンチマークには，バブルソート，階乗計算，素数計算を用いた（いずれも，コンパイル時の最適化は行っていない）．評価尺度としては，Issue Rate（1サイクルに発行する命令数）を用いる．

(1) 命令アラインメントおよびラインクロスの効果

以下をシミュレーションの前提とした．

- 分岐予測アルゴリズム：O型（初期状態：状態 '10'）
- 選択的命令無効化：有

シミュレーション結果を図7に示す．図7より，命令アラインメントを行った方が性能が良いことがわかる．また，ラインクロスの可／不可については，性能にほとんど影響を与えない．

(2) 分岐予測の効果

以下をシミュレーションの前提とした．

- 命令アラインメント：有り
- ラインクロス：不可
- 選択的命令無効化：無
- 分岐予測アルゴリズム（2-bit履歴）：O型（初期状態：状態 '10'）
- 分岐予測アルゴリズム（1-bit履歴）：分岐命令の前回の分岐結果がTAKENであればtaken予測，NOT-TAKENであればnot-taken予測

シミュレーション結果を図8に示す．図8からわかるように，分岐予測を行うことで，約2倍以上の性能向上が得られる．履歴のビット数については，2-bitの方がおおむね性能が良くなっているが大幅な性能向上ではない（ただし，階乗計算では1-bitの方が良い）．

(3) 選択的命令無効化の効果

以下をシミュレーションの前提とした．

- 命令アラインメント：有り
- ラインクロス：不可
- 分岐予測アルゴリズム（2-bit履歴）：Φ型，Z型，O型（初期

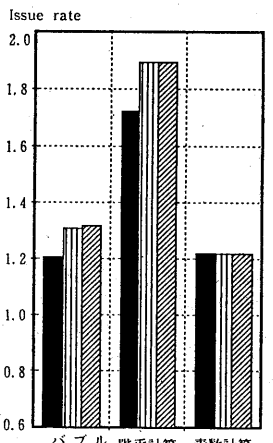

図7. アラインメントの効果

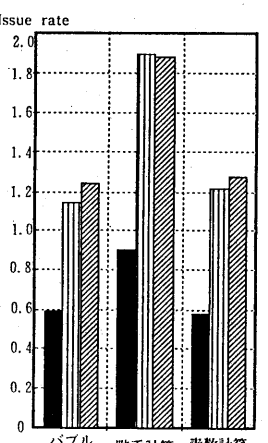

図8. 分岐予測の効果

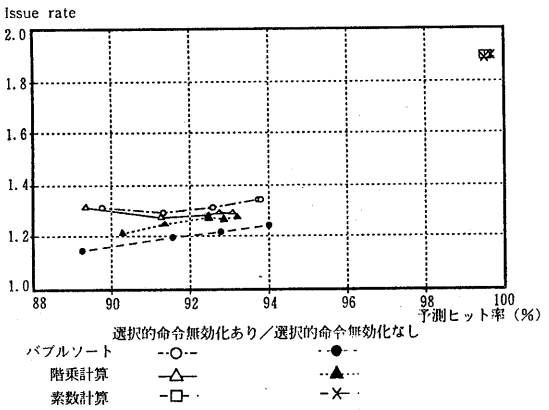

図9. 選択的命令無効化の効果

状態：状態 '10'）

- 分岐予測アルゴリズム（1-bit履歴）：分岐命令の前回の分岐結果がTAKENであればtaken予測，NOT-TAKENであればnot-taken予測

シミュレーション結果を図9に示す．グラフの横軸は分岐予測ヒット率であり，上記の4種の分岐予測アルゴリズムを用いた．図9から，選択的命令無効化を行った方がおおむね性能が良いことがわかる．また，選択的命令無効化を行わない場合は予測ヒット率が上がるにつれて性能も向上する傾向にあるが，一方，選択的命令無効化を行う場合は予測ヒット率に関係なく性能がほぼ一定である．よって，予測ヒット率が高い場合には選択的命令無効化の効果はさほどないが，予測ヒット率が低い場合は有効であることが分かる．

7. おわりに

以上，『新風』の命令供給機構について，要件，設計方針，および，構成と動作を述べた．本命令供給機構は，命令アラインメント，BTBによる分岐予測，および，選択的命令無効化といった特徴的な機能を有する．これらの機能の効果について，さらにシミュレーションによる性能評価を行った．

その中で最も重要な分岐予測については，選択的命令無効化との組合せにより，不規則な分岐パターンを多く含む（すなわち，予測ヒット率の低い）プログラムに対しても，ある程度の性能を保証できることを示した．

現在，ハードウェア開発と並行して，静的コード・スケジューリングを行う最適化コンパイラの開発を進めている．[8] [9] 最適化コンパイラにより，『新風』の分岐アーキテクチャの特長である先行条件決定方式 [7] を活かしたコード配置を行うことで，さらに分岐ペナルティの低減が図れるものと期待される．

謝辞

現在我々と共に『新風』の開発に携わっている，權 五鳳，入江直彦，弘中哲夫，音成 幹，納富 昭，岡崎恵三の各氏，および，日頃御討論頂く富田研究室の皆様に感謝致します．特に，入江直彦氏には，『新風』シミュレータによる性能測定において，多大なるご協力を頂いた．記して深謝します．

参考文献

[1] 村上ほか："SIMP（単一命令流／多重命令パイプライン）方式の構想，" 情報処理学会計算機アーキテクチャ研究会資料，88-CA-69-4（1988年1月）．

[2] 入江ほか："SIMP（単一命令流／多重命令パイプライン）方式に基づく『新風』プロセッサの高速化技法および性能予測，" 情報処理学会計算機アーキテクチャ研究会資料，88-ARC-73-11（1988年11月）．

[3] K.Murakami et al.："SIMP (Single Instruction Stream／Multiple Instruction Pipelining)：A Novel High-Speed Single-Processor Architecture,"Proc. 16th Int'l. Symp. Computer Architecture, pp.78-85, May 1989.

[4] 久我ほか："SIMP（単一命令流／多重命令パイプライン）方式に基づく『新風』プロセッサの低レベル並列処理アルゴリズム，" 情報処理学会論文誌，vol.30, no.12, pp.1603-1611（1989年12月）．

[5] 入江ほか："『新風』プロセッサにおける命令フェッチ機構，" 情報処理学会第38回全国大会講演論文集，5T-9（1989年3月）．

[6] 原ほか："『新風』プロセッサにおける分岐予測，" 情報処理学会第39回全国大会講演論文集，6X-5（1989年10月）．

[7] 原ほか："『新風』プロセッサの条件分岐方式，" 情報処理学会第40回全国大会投稿中．

[8] 入江ほか："SIMP（単一命令流／多重命令パイプライン）方式に基づくスーパースカラ・プロセッサ『新風』のための静的コード・スケジューリング技法，" 情報処理学会計算機アーキテクチャ研究会資料，89-ARC-79-6（1989年11月）．

[9] 入江ほか："統合型並列化コンパイラ・システム－局所コード・スケジューリング技法－，" 情報処理学会第40回全国大会投稿中．

[10] 原哲也："『新風』プロセッサの命令供給機構，" 九州大学工学部卒業論文（1989年2月）．

[11] W.G.Rosocha and E.S.Lee："Performance Enhancement of SISD Processors,"Proc. 6th Annual Symp. Computer Architecture, pp.216-231, Apr. 1979.

[12] J.E.Smith："A Study of Branch Prediction Strategies，" Proc. 8th Annual Symp. Computer Architecture, pp.135-148, May 1981.

[13] J.K.F.Lee and A.J.Smith："Branch Prediction Strategies and Branch Target Buffer Design,"IEEE Computer, vol.17, no.1, pp.6-22, Jan. 1984.

[14] S.McFarling and J.Hennessy："Reducing the Cost of Branches，" Proc. 13th Int'l. Symp. Computer Architecture, pp.396-403, June 1986.

[15] D.R.Ditzel and H.R.McLellan："Branch Folding in the CRISP Microprocessor：Reducing Branch Delay to Zero,"Proc. 14th Int'l. Symp. Computer Architecture, pp.2-9, June 1987.

[16] J.A.DeRosa and H.M.Levy："An Evaluation of Branch Architecture."Proc. 14th Int'l. Symp. Computer Architecture, pp.10-16, June 1987.

[17] A.R.Pleszkun et al.："WISQ：A Restartable Architecture Using Queues,"Proc. 14th Int'l. Symp. Computer Architecture, pp.290-299, June 1987.

[18] D.J.Lilja："Reducing the Branch Penalty in Pipelined Processors,"IEEE Computer, vol.21, no.7, pp. 47-55, July 1988.

[19] 馬場ほか："2レベルのキャッシュやパイプライン処理の工夫で速度を上げた大型コンピュータACOS1500，" 日経エレクトロニクス，no.373, pp.233-279（1985年7月）．

[20] 吉田ほか："パイプライン処理とブランチ命令，" 情報処理学会マイクロコンピュータ研究会資料，87-MC-44-1（1987年3月）．